Fermilab

TM-1095
2320.000

PASCAL AND FORTRAN COMPARISONS FOR THE PDP-11/34 AND VAX 11/780

A. M. Waller

March 1, 1982

I. Introduction

The Accelerator Control System's new host computer configuration employs a mixture of PDP-11/34's and VAX 11/780's. Most application programs will be written in a high level language for execution on the PDP-11/34. Two high level language compilers will be available to the application programmer for the development of programs targetted for the PDP-11/34. These will be FORTRAN IV-PLUS and OMSI PASCAL. Some applications will obviously be more attractive or convenient to code in FORTRAN, others in PASCAL. However, there will be a class of applications for which the "correct" or "best" choice of compiler is not obvious. In particular, there will be applications for which the execution speed and/or execution storage requirements are at a premium. In such cases it is necessary to know the relative properties of the target code produced by the two compilers.

Similar questions arise naturally for applications which are targetted for the VAX 11/780, where the choices of VAX-11 PASCAL and VAX-11 FORTRAN are available. This memo summarizes the results of tests that were made to compare the relative execution speeds and relative storage requirements of PASCAL and FORTRAN for analogous facilities (i.e. comparable arithmetic expressions, DO loops vs. WHILE loops, procedure calls, etc.). The results are collected under four separate headings:

- Efficiency comparisons of OMSI PASCAL and FORTRAN IV-PLUS
- Efficiency comparisons of VAX-11 PASCAL and VAX-11 FORTRAN
- Comments, suggestions and hints on using OMSI PASCAL in compatibility mode on the VAX 11/780 for programs tarrgetted for a PDP-11/34
- Comments, suggestions, and hints on using VAX-11 PASCAL for native mode programs

The comparisons described here should provide the application programmer with some guidance on some of the penalties he can expect to pay in using a particular compiler for many common operations. These comparisons do not claim to illustrate all the relevant differences between the compilers in question.

II. Comparison Methodology

The PASCAL and FORTRAN comparisons for the PDP-11/34 and the VAX 11/780 were done in three stages.

1. Cursory inspection of code generated for various common aspects of both languages.
2. WHETSTONE timing test.
3. Modified WHETSTONE timing program used to measure the non-weighted time of each module comprising the WHETSTONE test.

1.0 CURSORY INSPECTION

A non-runnable piece of coding was generated to inspect the following statements/expressions:

1. Arithmetic expressions with variables and constants.
   * Assignment expressions
   * Integer arithmetic (multiply, divide, subtract)
   * Exponentiation (not available in OMSI PASCAL)

2. Logical expressions.
   * IF-THEN-ELSE construct (only IF-THEN in FORTRAN IV-PLUS)
   * relational constructs

3. Loop expressions.
   * WHILE-DO/DO-WHILE (not available in FORTRAN IV PLUS)
   * FOR-TO-DO/DO

4. Procedure and function call statements.

5. CASE/(computed)GOTO statments.

6. Memory allocation of arrays.
   * BOOLEAN/LOGICAL*1
   * WORD/INTEGER*2
   * CHAR/CHARACTER (CHAR vs. BYTE for FORTRAN IV-PLUS)
   * REAL/REAL*4

2.0 WHETSTONE TIMING

The WHETSTONE performance test is a special "synthetic" benchmark program. It was designed to take into account machine architecture and compiler efficiency. Since the architecture of the machines remains the same, this test gives a good measure of PASCAL vs. FORTRAN compiler efficiency for the PDP-11/34 and then the VAX 11/780.

The benchmark reports results in Whetstone instructions/second. These "instructions" are only remotely connected with physical machine instructions (hence synthetic). The Whetstone instruction is a logical instruction such as "take integer result" or "goto" or a "call" block. These "instructions" may comprise several physical machine instructions. There are 11 modules in the program. Each module represents a typical, common group of statements that may well exist in a scientific program. To further simulate reality these modules are weighted based on statistics gathered about the style of scientific programming used at one institution.

The complete article and original ALGOL program which heralded the age of Whetstone performance measurements is available in the FERMILAB library. See below.

3.0 MODIFIED BENCHMARK

After looking at compiler generated code and Whetstone performance numbers, what else is there to do?

It would be interesting to see where the inefficiency of a compiler is when it is running the Whetstone test. The Whetstone program was modified to report timing values for each of the modules. This proved a useful tool in a few isolated cases where a particular loop timing for a compiler did not follow the general trend of timing differences. A look at the generated code revealed introductions of short cuts and other interesting side effects due to optimization.

III. Efficiency comparisons of OMSI PASCAL and FORTRAN IV-PLUS

The conclusions reached in this section are a result of the test programs that appear in Appendices A and B. The modified Whetstone test is not listed since it is almost the same coding as the Whetstone test program. For the modified Whetstone test; coding was inserted to measure the time it took to execute each of the major loops.

The second number in the FORTRAN IV-PLUS column in Table 1 below indicates the amount of coding generated with constants defined in PARAMETER statements instead of DATA statements.

PASCAL takes a few more bytes in order to store a value into the variable on every assignment statement. FORTRAN will leave values in registers until such time as it is needed to be stored into the variable. The philosophy of OMSI PASCAL is to disable variable assignment to registers in the main program if any external procedures are referenced (as in my test program). This is so because the compiler can not determine what variables may be used by such routines.

There are cases where the PARAMETER statement may cause more coding to be generated (as in the assignment block below). This is because PARAMETER forces the compiler to generate an instruction using a literal every time rather than optimizing on a register-to-register move. This is strictly a result of the compiler algorithms implemented to generate code. It serves only to show that an optimizing compiler can be "tricked" into "no-optimization" on some local instruction groups. In general, the results in savings using PARAMETER statements is examplified in the IF-THEN block below.

Although PARAMETER statements will generate literals, arithmetic expressions are not reduced during compile time. That is, the arithmetic coding is generated and executed at run time. OMSI PASCAL <u>does</u> reduce constants in arithmetic expressions at compile time (called constant folding). In the IF-THEN block below, where PASCAL is less efficient in code generation for assignment statements it makes up for in constant folding for a net gain on generated compiled code.

If the PASCAL procedure call is made and the argument value is in a register the number of bytes of code generated will be 2*N. If the argument value is stored then 4*N bytes are generated where N is the number of arguments. If the procedure is an external procedure only 2*N + 4 bytes of

coding are generated. If the procedure is NONPASCAL more interface code is generated and a jump to a FORTRAN interface routine is made. Thus the first set of numbers for the CALL statement under PASCAL are for the argument value being in the register. The second set is for stored argument values. The first number in each set is for a PASCAL external procedure. The second number is for a NON-PASCAL external procedure.

The computed GOTO for FORTRAN requires more overhead than a PASCAL statement.

A listing of the FORTRAN code used to produce Table 1 below appears in Appendix A.

TABLE 1
TABLE OF COMMON LANGUAGE ELEMENTS FOR OMSI PASCAL AND FORTRAN IV-PLUS

	PASCAL	FORTRAN IV-PLUS
ARITHMETIC AND ASSIGNMENT BLOCK	38 bytes	34/36 bytes
IF-THEN BLOCK	40 bytes	60/44 bytes
FOR-TO-DO/DO SHELL	12 bytes	12 bytes
CALL STATEMENT	2*N+4 2*N+18 bytes 4*N+4 4*N+18 bytes	8 bytes
CASE/(computed)GOTO SHELL	32 bytes	42 bytes

The only advantage of PASCAL PACKED arrays is in the TYPE BOOLEAN. For example; a PACKED array [1..8,1..32] OF BOOLEAN will allocate only 32 bytes of memory. The same DIMENSIONed array using LOGICAL*1 in FORTRAN will allocate 192 bytes of memory. Therefore PASCAL offers a 160 byte savings in this example.

Using the Whetstone program as a "typical" program; the following statistics were compiled from the load maps for total memory requirements.

The last three items in Table 2 below would need to have the word allocation modified in the Accelerator Control System. These items would be replaced by in-house written I/O for our real-time applications.

While not all of the initialization can go, such items as OPEN/CLOSE and common I/O routines may disappear or take on smaller procedure sizes.

For I/O format processing, PASCAL also includes the write integer and write real routines. Fortran appears to have read/write routines for integer, real, logical as complete packages. Thus if one is only writing integer in FORTRAN the package for reading as well as writing integer is loaded. In both PASCAL and FORTRAN only the necessary conversion routines are loaded. In both PASCAL and FORTRAN the general input and general output drivers are loaded.

For PASCAL, the common work area is basically HEAP/STACK storage. Although 2048 words were explicitly allocated here for HEAP storage, less could have been allocated. The 2048 words is a default minimum. The HEAP area will always need to be available for PASCAL if for nothing more than the stack. The size of the stack will vary from application to application and is principally determined by the maximum depth to which procedures are called. Heap is currently used for I/O control blocks as well as allocating dynamic memory when the procedure NEW is called. Heap is freed whenever a file is closed or whenever the procedure DISPOSE is called.

TABLE 2
LOAD MAP TABLE OF COMPARISONS FOR OMSI PASCAL AND FORTRAN IV-PLUS

	PASCAL	FORTRAN IV-PLUS
WHETSTONE CODE+DATA	1297 words	1284 words
MATH ROUTINES	401 words	505 words
INITIALIZATION OPEN/CLOSE COMMON I/O RTNS. ERROR REPORTING ROUTINES AND SUPPORT ROUTINES SUCH AS "SECND"	3776 words	2906 words
I/O FORMAT PROCESSING	1005 words	1387 words
COMMON WORK AREA LOGICAL UNIT TABLE FCS BUFFER AREA OBJ. TIME FORMAT BUFFER ETC.,ETC.	2144 words	1878 words

Table 3 below gives the modified and normal Whetstone benchmark times. The Whetstone benchmark indicates that OMSI PASCAL is approximately 16% less efficient than FORTRAN IV-PLUS.

Looking at the individual loop timings, only one loop time is grossly different. That one is in PASCAL loop number 9 with array referencing via procedure calls. (A listing of the PASCAL source code is given in Appendix B). This warrented an investigation.

It turns out that the two other local procedures used in the Whetstone test loops do not require full register use (registers R0-R5). One procedure uses the floating registers and the other only uses R3 and R5. In this one particular procedure registers R0-R3 are used for rapid indexing of array elements. A full context save of registers R0-R5 is done at the entry of this procedure and a restore is done at the end of the procedure. FORTRAN need not do this since it is not re-entrant. A quick check of the save and restore routines verifies that each time the procedure in loop number 9 is called there is an added overhead of +70 microseconds. The actual body of coding generated for the PASCAL procedure looks very similiar to the coding generated by the FORTRAN compiler (conceptually).

Loop number 8 in PASCAL is the second large variation and this is due to the fact that three arguments must be moved from local storage and moved to the stack before the procedure is called. This requires 12 bytes of coding for 3 move instructions so calling this procedure does take more overhead.

A listing of the PASCAL version of the Whetstone test appears in Appendix B.

TABLE 3

***** FORTRAN *****

```
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      256.68 MICRO-SEC.
 EXECUTION =     1612.00 MICRO-SEC.
 EXECUTION =       36.31 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      140.48 MICRO-SEC.
 EXECUTION =     3377.59 MICRO-SEC.
 EXECUTION =      145.42 MICRO-SEC.
 EXECUTION =       65.83 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      827.97 MICRO-SEC.
 ALL DONE
```

***** PASCAL *****

```
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      259.41 MICRO-SEC.
 EXECUTION =     1596.50 MICRO-SEC.
 EXECUTION =       33.51 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      154.44 MICRO-SEC.
 EXECUTION =     3260.42 MICRO-SEC.
 EXECUTION =      169.15 MICRO-SEC.
 EXECUTION =      145.02 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      860.44 MICRO-SEC.
 ALL DONE
```

****

```
LOOP1 = NIL
LOOP2 = ARRAY REFERENCING    IN-LINE
LOOP3 = ARRAY REFERENCING    ARRAY AS PROCEDURE PARAMETER
LOOP4 = CONDITIONAL JUMPS
LOOP5 = NIL
LOOP6 = INTEGER ARITHMETIC
LOOP7 = TRIG. FUNCTIONS
LOOP8 = PROCEDURE CALL
LOOP9 = ARRAY REFERENCING VIA PROCEDURE
LOOP10= NIL
LOOP11= STANDARD FUNCTIONS
```

TABLE 3 (continued)

```
***** BENCHMARK RESULTS *****

FORTRAN:

 BENCHMARK EXECUTION TIME IS    2.1411 MINUTES.
 EXECUTION =    233524.56 WHETSTONE INSTRUCTIONS/SEC.
 ALL DONE


PASCAL:

 BENCHMARK EXECUTION TIME IS    2.5442 MINUTES.
 EXECUTION =    196526.00 WHETSTONE INSTRUCTIONS/SEC.
 ALL DONE
```

IV. Efficiency comparisons of VAX PASCAL and VAX FORTRAN

The conclusions reached in this section are a result of the test programs that appear in Appendices C and D. The modified Whetstone test is not listed since it is almost the same coding as the Whetstone test program. For the modified Whetstone test, coding was inserted to measure the time it took to execute each of the major loops.

The second number in the two VAX FORTRAN columns in Table 4 below indicate the amount of coding generated with constants defined as PARAMETER statements instead of DATA statements.

As can be seen from Table 4 below, the PARAMETER statement can greatly reduce the amount of code generated. In all cases (optimized or non-optimized) VAX FORTRAN also does constant folding so that if constants appear in an arithmetic expression the compiler will reduce the result to a literal value. VAX PASCAL does not use constant folding. This is just the opposite of OMSI PASCAL and FORTRAN IV-PLUS for the PDP-11/34s. Optimization in VAX FORTRAN is for execution speed and the compiler will generate the necessary code to move values or addresses into registers before that code which comprises the body of the higher level language statement is executed. This can end up in the generation of more bytes of code as shown in the assignment block below.

Although execution is faster via register, the compiler mechanics implemented to generate the optimized code can be "tricked" into generating code which actually executes slower. This will be pointed out later. In general though the compiler optimization algorithms can realize an overall gain in efficiency of approximately 4%.

The formula given below for the amount of coding generated by a VAX PASCAL call statement is general and not totally applicable. The formula gives one a general idea on approximately how much code can be generated. It depends on how the compiler will generate coding (registers, memory reference, literals, etc.). However, +18 bytes are generated for every argument that is a string. This is because VAX PASCAL must set up the string descriptor on the stack before the procedure call. VAX FORTRAN just references a fixed list.

A listing of the PASCAL code used to produce Table 4 below appears in Appendix C.

TABLE 4
TABLE OF COMMON LANGUAGE ELEMENTS FOR VAX PASCAL AND VAX FORTRAN

	PASCAL	FORTRAN	
		optimized	non-optimized
ARITHMETIC AND ASSIGMENT BLOCK	47 bytes	50/21 bytes	44/17 bytes
IF-THEN-ELSE BLOCK	41 bytes	28/24 bytes	38/25 bytes
WHILE-DO/DO-WHILE SHELL	8 bytes	7 bytes	8 bytes
FOR-TO-DO/DO SHELL	15 bytes	15 bytes	13 bytes
CALL STATEMENT	see formula below	8 bytes	8 bytes
CASE/(computed)GOTO SHELL	28 bytes	27 bytes	27 bytes

N = 0 7 BYTES
N = 1 9 + 18*S BYTES
N > 1 ~ 12 + 6*(N-1) + 18*S BYTES

Where N is the number of parameters passed.
Where S is the number of string descriptor arguments passed.

The advantage of VAX PASCAL PACKED arrays is identical to the treatment given in the PDP-11/34 section of this paper.

The length of individual library sections (such as MATH, I/O, etc.) on a VAX VMS load map is impossible since the library is sharable. Table 5 is what can be shown for the Whetstone benchmark; again using this program as "typical".

TABLE 5
LOAD MAP TABLE OF COMPARISONS FOR VAX PASCAL AND VAX FORTRAN

	PASCAL	FORTRAN
WHETSTONE CODE+DATA	3901 bytes	2357 bytes
PAS$IOBASIC	2704 bytes	----------
PAS$IOOUTPUT	1330 bytes	----------
VIRTUAL MEMORY ALLOCATED	119296 bytes	113664 bytes

Table 6 below gives the modified and normal Whetstone benchmark times. The Whetstone benchmark indicates that VAX PASCAL is approximately 40% less efficient than optimized VAX FORTRAN.

Looking at individual loop timings, a passing comment will be made about loop number 8 in non-optimized VAX FORTRAN. (A listing of the FORTRAN source code is given in Appendix D). In this particular circumstance the optimizer which produced code to load variables into registers must, at the end of the routine, generate code to load the results back into memory locations. This takes extra code but, unfortunately, in this circumstance also extra time. The non-optimized code just generates memory referenced instructions so it need not go through the extra storage steps. Here the amount of code produced was small so the effects of memory vs. register speed is not well taken advantage of.

It will be noted that VAX PASCAL loop numbers 7, 8, 9, and 11 are markedly slower than VAX FORTRAN. All these loops make procedure calls. VAX FORTRAN takes advantage of its non-re-entrant nature and passes a local argument list via a CALLG instruction. There is no overhead of pushing arguments upon the stack before execution of the CALLG instruction. For math routines a special entry point is jumped to for VAX FORTRAN via the jump to subroutine instruction JSB thus reducing overhead even more. The loop timings show that this special VAX FORTRAN entry reduces overhead by about 3.5%. The inefficiency in VAX PASCAL is due to having to push arguments on the stack before using the CALLS instruction.

A listing of the FORTRAN version of the Whetstone test appears in Appendix D.

TABLE 6

****** OPTIMIZED FORTRAN ******

```
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       29.79 MICRO-SEC.
 EXECUTION =      235.88 MICRO-SEC.
 EXECUTION =        9.67 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       32.32 MICRO-SEC.
 EXECUTION =      296.30 MICRO-SEC.
 EXECUTION =       40.04 MICRO-SEC.
 EXECUTION =       27.19 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       88.72 MICRO-SEC.
 ALL DONE
```

****** NON- OPTIMIZED FORTRAN ******

```
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       33.48 MICRO-SEC.
 EXECUTION =      393.66 MICRO-SEC.
 EXECUTION =       11.90 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       39.80 MICRO-SEC.
 EXECUTION =      299.51 MICRO-SEC.
 EXECUTION =       38.24 MICRO-SEC.
 EXECUTION =       28.78 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       89.33 MICRO-SEC.
 ALL DONE
```

***** PASCAL NO OPTIMIZATION AVAILABLE ******

```
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       33.36 MICRO-SEC.
 EXECUTION =      281.32 MICRO-SEC.
 EXECUTION =       14.28 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =       39.25 MICRO-SEC.
 EXECUTION =      553.42 MICRO-SEC.
 EXECUTION =       70.83 MICRO-SEC.
 EXECUTION =       47.91 MICRO-SEC.
 EXECUTION =        0.00 MICRO-SEC.
 EXECUTION =      169.16 MICRO-SEC.
 ALL DONE
```

TABLE 6 (continued)

```
**************************************************************************

LOOP1 = NIL
LOOP2 = ARRAY REFERENCING    IN-LINE
LOOP3 = ARRAY REFERENCING    ARRAY AS PROCEDURE PARAMETER
LOOP4 = CONDITIONAL JUMPS
LOOP5 = NIL
LOOP6 = INTEGER ARITHMETIC
LOOP7 = TRIG. FUNCTIONS
LOOP8 = PROCEDURE CALL
LOOP9 = ARRAY REFERENCING VIA PROCEDURE
LOOP10= NIL
LOOP11= STANDARD FUNCTIONS


***** BENCHMARK RESULTS *****


OPTIMIZED FORTRAN:

 BENCHMARK EXECUTION TIME IS    0.9307 MINUTES.
 EXECUTION =   1146131.75 WHETSTONE INSTRUCTIONS/SEC.
 ALL DONE


NON-OPTIMIZED FORTRAN:

 BENCHMARK EXECUTION TIME IS    0.9650 MINUTES.
 EXECUTION =   1105354.00 WHETSTONE INSTRUCTIONS/SEC.
 ALL DONE


PASCAL:

 BENCHMARK EXECUTION TIME IS    1.5613 MINUTES.
 EXECUTION =    683176.75 WHETSTONE INSTRUCTIONS/SEC.
 ALL DONE
```

V. Comments on using OMSI PASCAL for the PDP-11/34

Some of the following comments are brought up in the OMSI PASCAL manual but are important enough to be re-iterated and expanded upon here.

The first thing to remember is to build OMSI PASCAL tasks with the FP and CP switches for floating point (we do have floating point) and to make the task checkpointable. If one does not make the task checkpointable one will specifically have to define the amount of dynamic heap storage through the linker (one must have sysgened their RSX system to include the Extend Task directive...as ours is).

The RESET and REWRITE procedures have three extra arguments. Other than the file variable argument there are file name, default file fields, and file size arguments. This is not standard but it is a very good extension to OMSI PASCAL for file definition.

The EXTERNAL directive is used for referencing other OMSI PASCAL procedures and functions. NONPASCAL is used for FORTRAN IV-PLUS and MACRO routines.

Since real precision is determined and set at initialization of the main program; all external modules must be compiled with the same real precision (single or double) as the main program.

Procedures and functions from one compilation form a single module and cannot be individually selected from the module. One should structure their modules wisely so that most or only the procedures needed will be linked.

The final point to make about modules involves the use of global. Since this topic applys to both OMSI PASCAL and VAX PASCAL it will be covered below in the VAX PASCAL section.

VI. Comments on using VAX PASCAL on the VAX 11/780

VAX PASCAL and OMSI PASCAL allow global sections in external modules to share the global in the main program. This makes all external modules appear as if they were compiled with the main program.

One should avoid global variables in modules, especially those modules which will be used by other people. While global is most tempting to use as a method of inter-procedure communication within a module; it is suggested that variables be passed as VAR parameters to other procedures.

If one must use global in external modules the type and size must correspond to the global existing in the main program. The use of these globals must be well understood by both main program and the external module.

APPENDIX A

```
      PROGRAM PASFOR
C     PARAMETER A=5.0,B=10.0,C=20.0,D=2.0
      BYTE PROMPT
      DATA A,B,C,D/5.0,10.0,20.0,2.0/
      Z=A
      X=A*B/C - Z
      Y=A*D
      Z = (C-B)+A
      IF((X.GE.Y) .AND. (Y.LT.C)) Z = (C+B)-A
      DO 10 I=1,100
               Z=Z-3.0
               X=X+1.5
10    CONTINUE
      PROMPT = '>'
      CALL PROC(PROMPT)
      MODE = 3
      GOTO(1,2,1,2,3,3) , MODE+1
      GOTO 4
1     X=0.0
      GOTO 4
2     Y=0.0
      GOTO 4
3     Z=0.0
4     CONTINUE
      END
```

APPENDIX B

```
PROGRAM WHETSTONE(OUTPUT,OUTFILE);
CONST    MAXWORD = 65535;
         START = 0;
         STOP = 1;
TYPE     RARRAY = ARRAY[1..4] OF REAL;
VAR      OUTFILE: TEXT;
         X1,X2,X3,X4,X,Y,Z,T,T1,T2: REAL;
         E1: RARRAY;
         I,J,K,L,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,N11: INTEGER;
         LOOP,JJ: INTEGER;
         STIME,TTIME: REAL;
   WALL,WALLO: REAL;
FUNCTION WTIME( MODE:INTEGER ): REAL;
VAR ZERO: REAL;
FUNCTION SECNDS(VAR PARAM: REAL): REAL;NONPASCAL;
BEGIN
ZERO := 0.0;
IF MODE <> 0 THEN
   BEGIN
    WALL := SECNDS(WALLO);
    WTIME := WALL;
    WALLO := SECNDS(ZERO);
   END
ELSE
   BEGIN
    WALLO := SECNDS(ZERO);
    WTIME := ZERO;
   END;
END;
PROCEDURE PA( VAR E:RARRAY );
VAR      J: INTEGER;
BEGIN
   J := 0;
   WHILE J<6 DO
      BEGIN
         E[1] := (E[1]+E[2]+E[3]-E[4])*T;
         E[2] := (E[1]+E[2]-E[3]+E[4])*T;
         E[3] := (E[1]-E[2]+E[3]+E[4])*T;
         E[4] := (-E[1]+E[2]+E[3]+E[4])/T2;
         J := J+1;
      END;
END      (* PROCEDURE PA *);
PROCEDURE P0;
BEGIN
   E1[J] := E1[K];
   E1[K] := E1[L];
   E1[L] := E1[J];
END      (* PROCEDURE P0 *);
PROCEDURE P3( VAR X,Y,Z:REAL);
VAR X1,Y1: REAL;
BEGIN
   X1 := X;
   Y1 := Y;
   X1 := T*(X1+Y1);
```

```
   Y1 := T*(X1+Y1);
   Z := (X1+Y1)/T2;
END      (* PROCEDURE P3 *);
PROCEDURE POUT( VAR N,J,K:INTEGER ; VAR X1,X2,X3,X4:REAL);
BEGIN
   WRITELN(OUTFILE,N:7,J:7,K:7,X1:12:4,X2:12:4,X3:12:4,X4:12:4);
END      (* PROCEDURE POUT *);
BEGIN    (* WHETSTONE.PAS *)
(* SPECIAL MODIFICATION OF REWRITE FOR OMSI PASCAL ONLY!!! *)
   REWRITE(OUTPUT,'PASRPT','.DAT');
   REWRITE(OUTFILE,'PASWHT','.DAT');
   WALL := 0.0;
   WALL0 := 0.0;
(* IF LOOP = 10 THEN THIS CORRESPONDS TO THE EXECUTION OF
   10 TIMES 1,000,000 WHETSTONE INSTRUCTIONS.                    *)
   LOOP := 30;
   STIME := WTIME(START);
   T := 0.499975;
   T1 := 0.50025;
   T2 := 2.0;
(* I = 10 CORRESPONDS TO 1,000,000 WHETSTONE INSTRUCTIONS/MAJOR LOOP
*)
   I := 10;
(* ESTABLISH MODULE EXECUTION TIME PARAMETERS      *)
   N1 := 0;
   N2 := 12*I;
   N3 := 14*I;
   N4 := 345*I;
   N5 := 0;
   N6 := 210*I;
   N7 := 32*I;
   N8 := 899*I;
   N9 := 616*I;
   N10 := 0;
   N11 := 93*I;
(* BEGINNING OF MAJOR LOOP
   WE HAVE THIS MAJOR LOOP RATHER THAN JUSING A BIGGER I BECAUSE
   A LARGER I (GREATER THAN 36) WILL OVERFLOW ON A 16 BIT MACHINE  *)
   FOR JJ:=1 TO LOOP DO
      BEGIN
(* MODULE 1 : SIMPLE IDENTIFIERS *)
         X1 := 1.0;
         X2 := -1.0;
         X3 := -1.0;
         X4 := -1.0;
         FOR I:=1 TO N1 DO
            BEGIN
               X1 := (X1+X2+X3-X4)*T;
               X2 := (X1+X2-X3+X4)*T;
               X3 := (X1-X2+X3+X4)*T;
               X4 := (-X1+X2+X3+X4)*T;
            END (* MODULE 1 *);
         IF (JJ=LOOP)
            THEN
```

```
                POUT(N1,N1,N1,X1,X2,X3,X4);
(* MODULE 2 :  ARRAY ELEMENTS *)
          E1[1] := 1.0;
          E1[2] := -1.0;
          E1[3] := -1.0;
          E1[4] := -1.0;
          FOR I:=1 TO N2 DO
             BEGIN
                E1[1] := (E1[1]+E1[2]+E1[3]-E1[4])*T;
                E1[2] := (E1[1]+E1[2]-E1[3]+E1[4])*T;
                E1[3] := (E1[1]-E1[2]+E1[3]+E1[4])*T;
                E1[4] := (-E1[1]+E1[2]+E1[3]+E1[4])*T;
             END (* MODULE 2 *);
          IF (JJ=LOOP)
             THEN
                POUT(N2,N3,N2,E1[1],E1[2],E1[3],E1[4]);
(* MODULE 3 :  ARRAY AS PARAMETER *)
          FOR I:=1 TO N3 DO
             PA(E1);
(* END MODULE 3 *)
          IF (JJ=LOOP)
             THEN
                POUT(N3,N2,N2,E1[1],E1[2],E1[3],E1[4]);
(* MODULE 4 :  CONDITIONAL JUMPS *)
          J := 1;
          FOR I:=1 TO N4 DO
             BEGIN
                IF J=1
                   THEN
                      J := 2
                   ELSE
                      J := 3;
                IF J>2
                   THEN
                      J := 0
                   ELSE
                      J := 1;
                IF J<1
                   THEN
                      J := 1
                   ELSE
                      J := 0;
             END (* MODULE 4 *);
          IF (JJ=LOOP)
             THEN
                POUT(N4,J,J,X1,X2,X3,X4);
(* MODULE 5 :  OMITTED *)
(* MODULE 6 :  INTEGER ARITHMETIC *)
          J := 1;
          K := 2;
          L := 3;
          FOR I:=1 TO N6 DO
             BEGIN
                J := J*(K-J)*(L-K);
```

```
                K := L*K-(L-J)*K;
                L := (L-K)*(K+J);
                E1[L-1] := J+K+L;
                E1[K-1] := J*K*L;
             END (* MODULE 6 *);
          IF (JJ=LOOP)
             THEN
                POUT(N6,J,K,E1[1],E1[2],E1[3],E1[4]);
(* MODULE 7 :  TRIG.  FUNCTIONS *)
          X := 0.5;
          Y := 0.5;
          FOR I:=1 TO N7 DO
             BEGIN
                X :=
T*ARCTAN(T2*SIN(X)*COS(X)/(COS(X+Y)+COS(X-Y)-1.0));
                Y :=
T*ARCTAN(T2*SIN(Y)*COS(Y)/(COS(X+Y)+COS(X-Y)-1.0));
             END (* MODULE 7 *);
          IF (JJ=LOOP)
             THEN
                POUT(N7,J,K,X,X,Y,Y);
(* MODULE 8 :  PROCEDURE CALLS *)
          X := 1.0;
          Y := 1.0;
          Z := 1.0;
          FOR I:=1 TO N8 DO
             P3(X,Y,Z);
(* END MODULE 8 *)
          IF (JJ=LOOP)
             THEN
                POUT(N8,J,K,X,Y,Z,Z);
(* MODULE 9 :  ARRAY REFERENCES *)
          J := 1;
          K := 2;
          L := 3;
          E1[1] := 1.0;
          E1[2] := 2.0;
          E1[3] := 3.0;
          FOR I:=1 TO N9 DO
             P0;
(* END MODULE 9 *)
          IF (JJ=LOOP)
             THEN
                POUT(N9,J,K,E1[1],E1[2],E1[3],E1[4]);
(* MODULE 10 :  INTEGER ARITHMETIC *)
          J := 2;
          K := 3;
          FOR I:=1 TO N10 DO
             BEGIN
                J := J+K;
                K := J+K;
                J := K-J;
                K := K-J-J;
             END (* MODULE 10 *);
```

```
          IF (JJ=LOOP)
             THEN
                POUT(N10,J,K,X1,X2,X3,X4);
(* MODULE 11 : STANDARD FUNCTIONS *)
          X := 0.75;
          FOR I:=1 TO N11 DO
             X := SQRT(EXP(LN(X)/T1));
(* END MODULE 11 *)
          IF (JJ=LOOP)
             THEN
                POUT(N11,J,K,X,X,X,X);
       END;
    TTIME := WTIME(STOP);
    STIME := TTIME;
    TTIME := TTIME/60.0;
    STIME := LOOP*1.0E6/STIME;
    WRITELN(' BENCHMARK EXECUTION TIME IS ',TTIME:8:4,' MINUTES.');
    WRITELN(' EXECUTION = ',STIME:11:2,' WHETSTONE INSTRUCTIONS/SEC.');
    WRITELN(' ALL DONE ');
END.
```

APPENDIX C

```
PROGRAM FORPAS;
CONST
       A = 5.0;
       B = 10.0;
       C = 20.0;
       D = 2.0;
       %INCLUDE 'SYS$LIBRARY:LIBDEF.PAS/NOLIST'
TYPE
      WORDLEN = 0..65535;
      WORD = PACKED RECORD
                 WORDTYPE: WORDLEN
             END;
      STRINGLEN = 1..72;
      STRING = PACKED ARRAY [STRINGLEN] OF CHAR;
      MESSGLEN = 1..12;
      MESSAGE = PACKED ARRAY [MESSGLEN] OF CHAR;
VAR
     X,Y,Z: REAL;
     I: INTEGER;
     MODE: INTEGER;
     COMMANDLINE: STRING;
     STATUS: INTEGER;
     COMLEN: WORD;
     PROMPT: MESSAGE;
FUNCTION LIB$GETFOREIGN(%STDESCR COM:STRING;
                        %STDESCR PRMPT:MESSAGE;
                        VAR LEN:WORD) : INTEGER;
EXTERN;
PROCEDURE LIB$STOP(%IMMED ERROR:INTEGER)
EXTERN;
BEGIN
    Z := A;
    X := A*B/C-Z;
    Y := A**D;
    IF (X >= Y) AND (Y <C)
       THEN
          Z := (C+B)-A
       ELSE
          Z := (C-B)+A;
    I := 0;
    WHILE I < 20 DO
       BEGIN
          I := I+1;
          Z := Z+2.0;
       END;
    FOR I:=1 TO 100 DO
       BEGIN
          Z := Z-3.0;
          X := X+1.5;
       END;
    PROMPT := 'FILE NAMES:
    STATUS := LIB$GETFOREIGN(COMMANDLINE,PROMPT,COMLEN);
    IF STATUS <> LIB$NORMAL
       THEN
```

```
        LIB$STOP(STATUS)
    MODE := 3;
    CASE MODE OF
       0,2:  X := 0.0;
       1,3:  Y := 0.0;
       4,5:  Z := 0.0;
    END;
END.
```

APPENDIX D

```
      DIMENSION E1(4)
      COMMON T,T1,T2,E1,J,K,L
C
C     SET NTTY FOR TERMINAL OUTPUT DEVICE NUMBER
C
      NTTY = 7
C
C     IF LOOP = 10 THEN THIS CORRESPONDS TO THE EXECUTION OF
C     10 TIMES 1,000,000 WHETSTONE INSTRUCTIONS.
C
      LOOP = 64
C
      STIME = BTIME(0)
      T = 0.499975
      T1 = 0.50025
      T2 = 2.0
C
C     I = 10 CORRESPONDS TO 1,000,000 WHETSTONE INSTRUCTIONS/MAJOR LOOP
C
      I = 10
C
C     ESTABLISH MODULE EXECUTION TIME PARAMETERS
C
      N1 = 0
      N2 = 12*I
      N3 = 14*I
      N4 = 345*I
      N5 = 0
      N6 = 210*I
      N7 = 32*I
      N8 = 899*I
      N9 = 616*I
      N10 = 0
      N11 = 93*I
C
C     BEGINNING OF MAJOR LOOP
C     WE HAVE THIS MAJOR LOOP RATHER THAN USING A BIGGER I BECAUSE
C     A LARGER I (GREATER THAN 36) WILL OVERFLOW ON A 16 BIT MACHINE.
C
      DO 500 JJ = 1,LOOP
C
C     BEGINNING OF MODULE 1, SIMPLE IDENTIFIERS
C
      X1 = 1.0
      X2 = -1.0
      X3 = -1.0
      X4 = -1.0
      IF(N1) 19,19,11
11    DO 18 I = 1,N1,1
                X1 = (X1+X2+X3-X4)*T
                X2 = (X1+X2-X3+X4)*T
                X3 = (X1-X2+X3+X4)*T
                X4 = (-X1+X2+X3+X4)*T
18    CONTINUE
```

```
19 CONTINUE
   IF(JJ.EQ.LOOP) CALL POUT(N1,N1,N1,X1,X2,X3,X4)
C
C  BEGINNING OF MODULE 2, ARRAY ELEMENTS
C
   E1(1) = 1.0
   E1(2) = -1.0
   E1(3) = -1.0
   E1(4) = -1.0
   IF(N2) 29,29,21
21 DO 28 I = 1,N2,1
           E1(1) = (E1(1)+E1(2)+E1(3)-E1(4))*T
           E1(2) = (E1(1)+E1(2)-E1(3)+E1(4))*T
           E1(3) = (E1(1)-E1(2)+E1(3)+E1(4))*T
           E1(4) = (-E1(1)+E1(2)+E1(3)+E1(4))*T
28 CONTINUE
29 CONTINUE
   IF(JJ.EQ.LOOP) CALL POUT(N2,N3,N2,E1(1),E1(2),E1(3),E1(4))
C
C  BEGINNING OF MODULE 3, ARRAY AS A PARAMETER
C
   IF(N3) 39,39,31
31 DO 38 I = 1,N3,1
38 CALL PA(E1)
39 CONTINUE
   IF(JJ.EQ.LOOP) CALL POUT(N3,N2,N2,E1(1),E1(2),E1(3),E1(4))
C
C  BEGINNING OF MODULE 4, CONDITIONAL JUMPS
C
   J = 1
   IF(N4) 49,49,41
41 DO 48 I = 1,N4,1
           IF(J-1) 43,42,43
42         J = 2
           GO TO 44
43         J = 3
44         IF(J-2) 46,46,45
45         J = 0
           GO TO 47
46         J = 1
47         IF(J-1) 411,412,412
411                J = 1
           GO TO 48
412                J = 0
48 CONTINUE
49 CONTINUE
   IF(JJ.EQ.LOOP) CALL POUT(N4,J,J,X1,X2,X3,X4)
C
C  THERE IS NO MODULE 5
C
C
C  BEGINNING OF MODULE 6, INTEGER ARITHMETIC
C
   J = 1
```

```
      K = 2
      L = 3
      IF(N6) 69,69,61
61    DO 68 I = 1,N6,1
                J = J*(K-J)*(L-K)
                K = L*K-(L-J)*K
                L = (L-K)*(K+J)
                E1(L-1) = J+K+L
                E1(K-1) = J*K*L
68    CONTINUE
69    CONTINUE
      IF(JJ.EQ.LOOP) CALL POUT(N6,J,K,E1(1),E1(2),E1(3),E1(4))
C
C     BEGINNING OF MODULE 7,  TRIG.  FUNCTIONS
C
      X = 0.5
      Y = 0.5
      IF(N7) 79,79,71
71    DO 78 I = 1,N7,1
                X = T*ATAN(T2*SIN(X)*COS(X) / (COS(X+Y)+COS(X-Y)-1.0))
                Y = T*ATAN(T2*SIN(Y)*COS(Y) / (COS(X+Y)+COS(X-Y)-1.0))
78    CONTINUE
79    CONTINUE
      IF(JJ.EQ.LOOP) CALL POUT(N7,J,K,X,X,Y,Y)
C
C     BEGINNING OF MODULE 8,  PROCEDURE CALLS
C
      X = 1.0
      Y = 1.0
      Z = 1.0
      IF(N8) 89,89,81
81    DO 88 I = 1,N8,1
88    CALL P3(X,Y,Z)
89    CONTINUE
      IF(JJ.EQ.LOOP) CALL POUT(N8,J,K,X,Y,Z,Z)
C
C     BEGINNING OF MODULE 9,  ARRAY REFERENCES
C
      J = 1
      K = 2
      L = 3
      E1(1) = 1.0
      E1(2) = 2.0
      E1(3) = 3.0
      IF(N9) 99,99,91
91    DO 98 I = 1,N9,1
98    CALL P0
99    CONTINUE
      IF(JJ.EQ.LOOP) CALL POUT(N9,J,K,E1(1),E1(2),E1(3),E1(4))
C
C     BEGINNING OF MODULE 10,  INTEGER ARITHMETIC
C
      J = 2
      K = 3
```

```
      IF(N10) 109,109,101
101   DO 108 I = 1,N10,1
      J = J+K
      K = J+K
      J = K-J
      K = K -J-J
108   CONTINUE
109   CONTINUE
      IF(JJ.EQ.LOOP) CALL POUT(N10,J,K,X1,X2,X3,X4)
C
C  BEGINNING OF MODULE 11, STANDARD FUNCTIONS
C
      X = 0.75
      IF(N11) 119,119,111
111   DO 118 I = 1,N11,1
118   X = SQRT(EXP(ALOG(X)/T1))
119   CONTINUE
      IF(JJ.EQ.LOOP) CALL POUT(N11,J,K,X,X,X,X)
C
C  THIS IS THE END OF THE MAJOR LOOP
C
500   CONTINUE
C
C  NOW PRINT THE EXECUTION TIME
C
      TTIME = BTIME(1)
      STIME = TTIME
      TTIME = TTIME/60.0
      STIME = LOOP*1.0E6/STIME
      WRITE(NTTY,1000) TTIME
1000  FORMAT(' BENCHMARK EXECUTION TIME IS ',F8.4,' MINUTES.')
      WRITE(NTTY,1001) STIME
1001  FORMAT(' EXECUTION = ',F11.2,' WHETSTONE
INSTRUCTIONS/SEC.')
      WRITE(NTTY,1002)
1002  FORMAT(9H ALL DONE,/)
      STOP
      END
      SUBROUTINE PA(E)
      COMMON T,T1,T2
      DIMENSION E(4)
      J = 0
1     CONTINUE
      E(1) = (E(1)+E(2)+E(3)-E(4))*T
      E(2) = (E(1)+E(2)-E(3)+E(4))*T
      E(3) = (E(1)-E(2)+E(3)+E(4))*T
      E(4) = (-E(1)+E(2)+E(3)+E(4))/T2
      J = J+1
      IF(J-6) 1,2,2
2     CONTINUE
      RETURN
      END
      SUBROUTINE P0
      COMMON T,T1,T2,E1(4),J,K,L
```

```
      E1(J) = E1(K)
      E1(K) = E1(L)
      E1(L) = E1(J)
      RETURN
      END
      SUBROUTINE P3(X,Y,Z)
      COMMON T,T1,T2
      X1 = X
      Y1 = Y
      X1 = T*(X1+Y1)
      Y1 = T*(X1+Y1)
      Z = (X1+Y1)/T2
      RETURN
      END
      SUBROUTINE POUT(N,J,K,X1,X2,X3,X4)
      WRITE(6,1) N,J,K,X1,X2,X3,X4
1     FORMAT(1H ,3I7,4E12.4)
      RETURN
      END
```

Distribution

ACNET Design Group
Controls Software Staff
R. Ducar MS307
W. Knopf MS307
J. Zagel MS307
J. Tinsley MS306
file
J.F. Bartlett MS222
D. Ritchie MS120

amw: USR$DISK:[WALLER.TEXT]PASREPORT.RNO